Design and Implementation of a high performance IPC for Intrusion Prevention using Socket API

Bachelorthesis

Daniel von Rauchhaupt

Universtität Potsdam
Institut für Informatik und Computational Science
Professur Betriebssysteme und Verteilte Systeme

July 31, 2024

Agenda

Motivation

Motivation

Host-based intrusion detection and prevention

Threats:

- access data,
- manipulate data, or
- render a system unreliable or unusable.

Host-based intrusion detection and prevention

Necessity for Intrusion Prevention Systems:

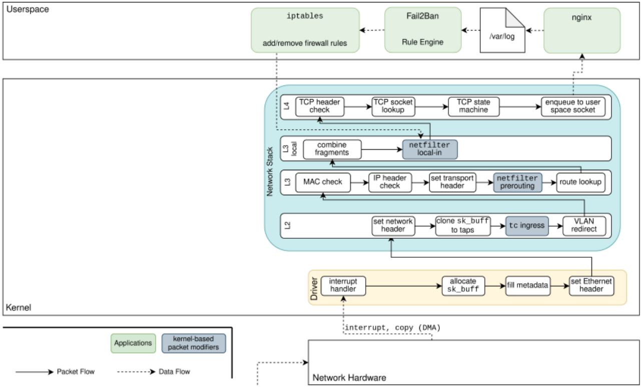
1. The majority of systems have vulnerabilities, rendering them susceptible.
2. Replacing systems with known vulnerabilities is difficult. Specific features may only be present in the less-secure system.
3. Developing absolutely secure systems is difficult, since the explicit absence of vulnerabilities can rarely be proven.
4. Secure systems remain vulnerable to insiders misusing their privileges.

Motivation - Fail2ban

Fail2ban application creates "jails":

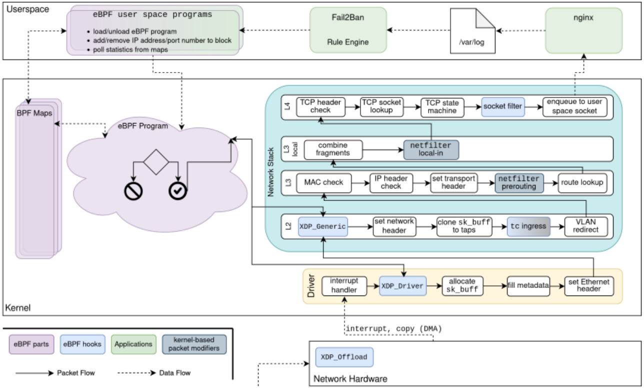
1. A jail consists out of:
    - Log path
    - Specific filter (uses Regex)
    - A defined action
    - Multiple customizable parameters (Ban duration, Ban limit)
2. Jails are saved on persistent storage
3. Deduces vital client information from log messages

Motivation - Fail2ban



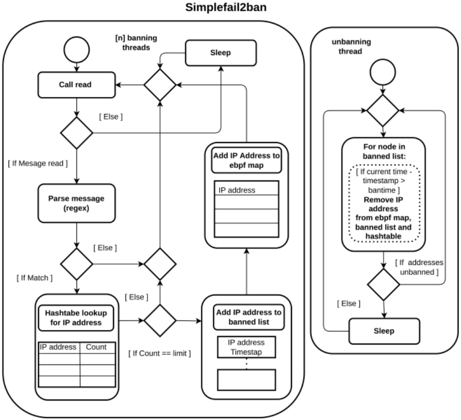
Florian Mikolajczak: Implementation and Evaluation of an Intrusion Prevention System Leveraging eBPF on the Basis of Fail2Ban

Motivation - Fail2ban



Florian Mikolajczak: Implementation and Evaluation of an Intrusion Prevention System Leveraging eBPF on the Basis of Fail2Ban

Motivation - Fail2ban



Paul Raatschen: Design and Implementation of a new Inter-Process Communication Architecture for Log-based HIDS for 100 GbE Environments

Design

Design

UNIX domain sockets

An alternative to the shared memory mode: UNIX domain Sockets

1. Preferred over internet sockets
2. Three types of UNIX domain sockets:
   - SOCK_STREAM: Stream-oriented socket. Establishes connections and keeps them open until explicitly closed.
   - SOCK_DGRAM: Datagram-oriented socket. Preserves message boundaries. Mostly reliable.
   - SOCK_SEQPACKET: Sequence-packet socket. Is connection-oriented, preserves message boundaries, and retains the order in which data was sent.

UNIX domain sockets

An alternative to the shared memory mode: UNIX domain Sockets

1. Preferred over internet sockets
2. Three types of UNIX domain sockets:
   - SOCK_STREAM: Stream-oriented socket. Establishes connections and keeps them open until explicitly closed.
   - SOCK_DGRAM: Datagram-oriented socket. Preserves message boundaries. Mostly reliable.
   - SOCK_SEQPACKET: Sequence-packet socket. Is connection-oriented, preserves message boundaries, and retains the order in which data was sent.

Block

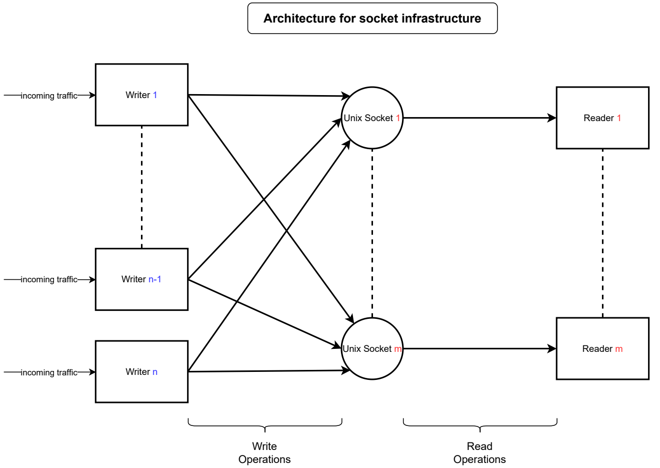
-> SOCK_SEQPACKET is preferred

UNIX domain sockets

An alternative to the shared memory mode:

1. Existing support on all UNIX systems
2. Established Write and Read API
3. Kernel-based IPC promising low latency and high bandwidth
4. Easily scalable beyond the local system

UNIX domain sockets



Implementation

Implementation

Shared parameters

Shared parameters:

```c
#define MAX_AMOUNT_OF_SOCKETS 32
#define SOCKET_TEMPLATE_LENGTH 128
#define SOCKET_NAME_TEMPLATE "/tmp/unixDomainSock4SF2B_"
```

Union defining which process is calling a function:

```c
union sock_arg_t{
    struct sock_writer_arg_t wargs;
    struct sock_reader_arg_t rargs;
};
```

Auxiliary functions

Initialization of socket IPC:

```
int sock_init(
    union sock_arg_t *sock_args,
    int role
);
```

Cleanup of socket IPC:

```
int sock_cleanup(
    union sock_arg_t *sock_args,
    int role
);
```

Write API

Writer structure:

```
struct sock_writer_arg_t
{
        char socketPathNames
              [MAX_AMOUNT_OF_SOCKETS][SOCKET_TEMPLATE_LENGTH];
        struct sockaddr_un
              socketConnections[MAX_AMOUNT_OF_SOCKETS];
        int socketRecvs[MAX_AMOUNT_OF_SOCKETS];
        int writeSockets[MAX_AMOUNT_OF_SOCKETS];
};
```

Write API

Write function:

```
int sock_writev(
        struct sock_writer_arg_t *sock_args,
        struct iovec *log_iovs,
        uint16_t invalid_count,
        uint16_t maxNumOfSocks
);
```

Read API

Reader structure:

```
struct sock_reader_arg_t
{
    char socketPathName[SOCKET_TEMPLATE_LENGTH];
    struct sockaddr_un address;
    int sizeOfAddressStruct;
    int readSocket;
    int clientSockets[MAX_AMOUNT_OF_SOCKETS];
};
```

Read API

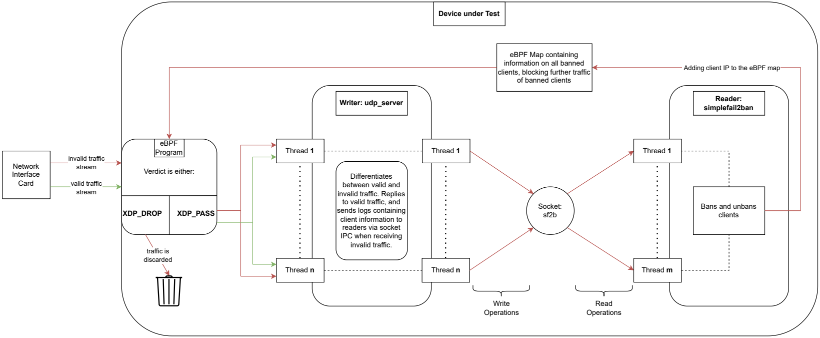
Read function:

```c
int sock_readv(
    struct sock_reader_arg_t *sock_args,
    struct iovec *iovecs
);
```

Experiments

Experiments

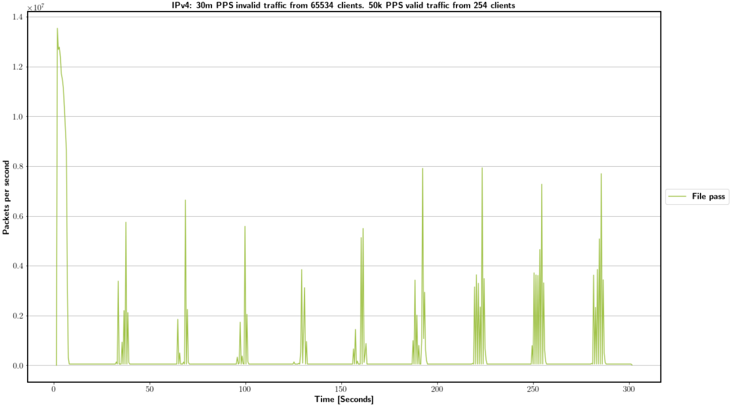
Device under Test


Device under Test
eBPF Map containing information on all banned clients, blocking further traffic of banned clients
Adding client IP to the eBPF map
Writer: udp_server
Reader: simplefail2ban
Network Interface Card
invalid traffic stream
valid traffic stream
eBPF Program
Verdict is either:
XDP_DROP
XDP_PASS
traffic is discarded
Thread 1
Thread n
Differentiates between valid and invalid traffic. Replies to valid traffic, and sends logs containing client information to readers via socket IPC when receiving invalid traffic.
Thread 1
Thread n
Socket: sf2b
Thread 1
Thread m
Bans and unbans clients
Write Operations
Read Operations

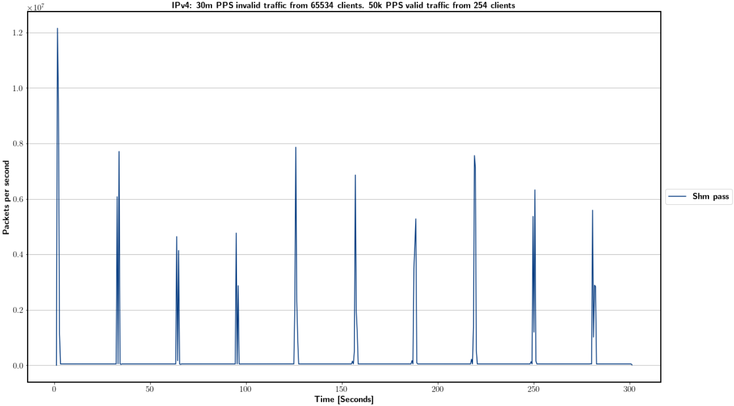
Factors and their levels

1. IP stack: IPv4, IPv6 and IPv4/IPv6 mixed
2. Effects of differing amount of invalid traffic sent: 100k, 1M, 10M, 20M, 30M PPS
3. Effects of differing number of clients sending invalid data: 65,534 (from 256 subnets) and 131,068 (from 512 subnets)
4. Differing IPC type: FILE (traditional file-based logging), SHM (using shared memeory), SOCK (using UNIX domain sockets)
    - If applicable: No 2nd Reader/ Enabling 2nd Reader

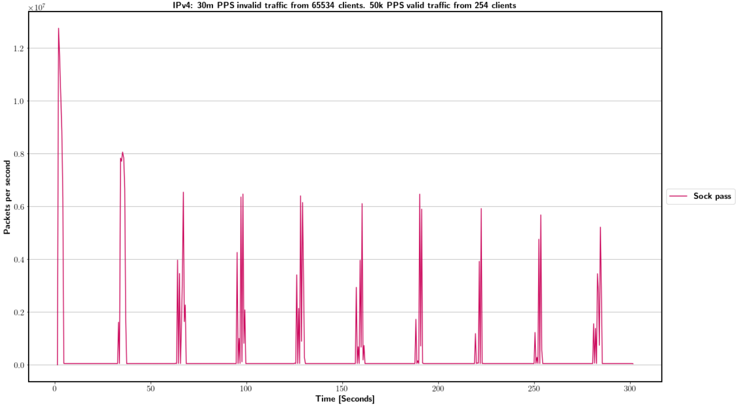
File pass: IPv4 - 65534 Clients - 30M invalid PPS



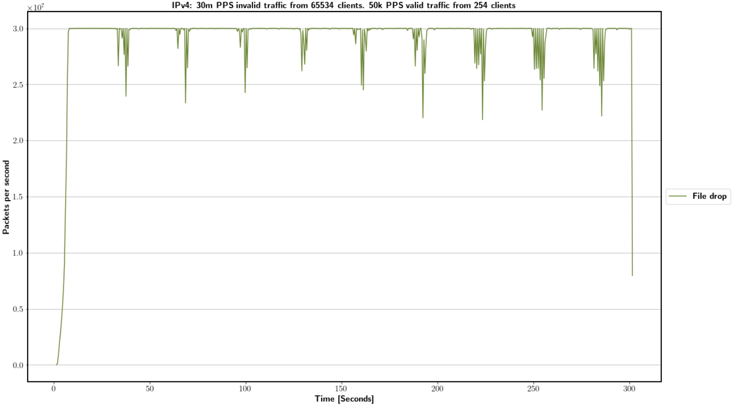
Shm pass: IPv4 - 65534 Clients - 30M invalid PPS



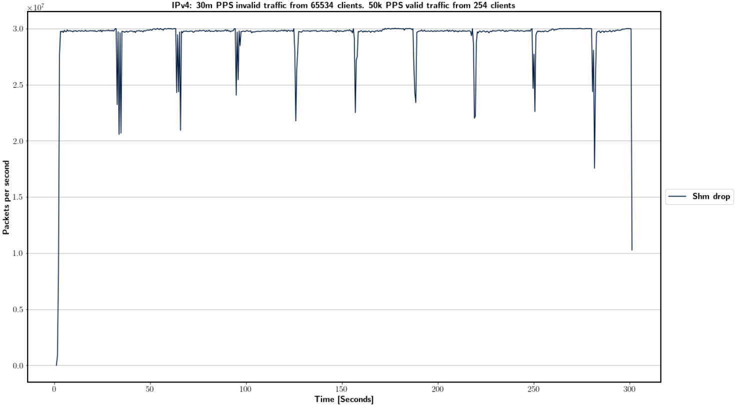
Sock pass: IPv4 - 65534 Clients - 30M invalid PPS



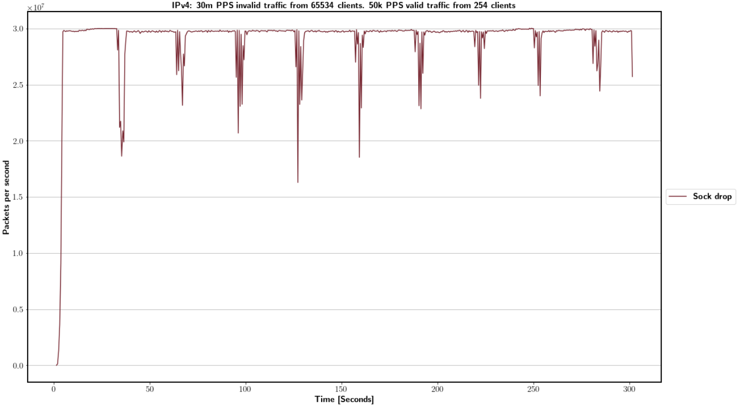
File drop: IPv4 - 65534 Clients - 30M invalid PPS



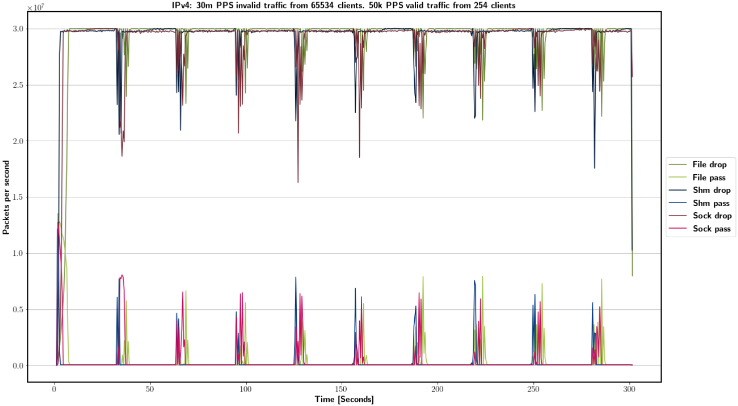
Shm drop: IPv4 - 65534 Clients - 30M invalid PPS



Sock drop: IPv4 - 65534 Clients - 30M invalid PPS



IPv4 - 65534 Clients - 30M invalid PPS - 50k valid PPS



IPv4 - 65534 Clients - 30M invalid PPS - 50k valid PPS

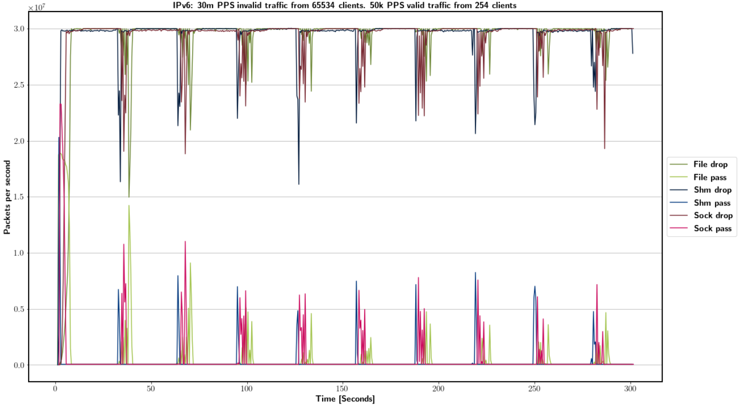
IPC type	XDP_DROP [10^8]	XDP_PASS [10^6]	Relative drop [%]
File	87.75	159.82	97.52375345
Shm	88.30	87.23	98.13105047
Sock	87.45	139.42	97.18179422

IPC type	Packets received by udp_server [10^6]	Log messages [10^6]	CPU [seconds]
File	17.48	4.07	16.55
Shm	21.39	6.99	39.08
Sock	16.92	3.16	138.85

Block

Total packets sent: 9,015m. Best-case drop rate: 99.97815533%

IPv6 - 65534 Clients - 30M invalid PPS - 50k valid PPS



IPv6 - 65534 Clients - 30M invalid PPS - 50k valid PPS

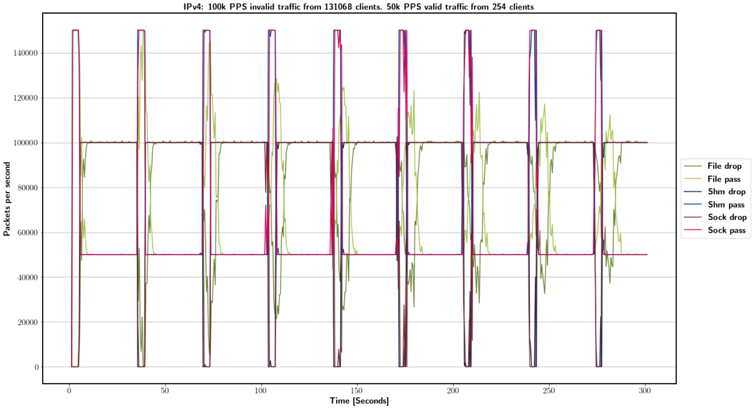
IPC type	**XDP_DROP [10^8]**	**XDP_PASS [10^6]**	**Relative drop [%]**
File	87.41	211.05	97.14091697
Shm	88.63	85.55	98.50239609
Sock	87.77	170.03	97.54838057

IPC type	**Packets received by udp_server [10^6]**	**Log messages [10^6]**	**CPU [seconds]**
File	17.20	3.87	22.51
Shm	21.79	7.24	46.03
Sock	16.92	3.00	149.69

Block

Total packets sent: 9,015m. Best-case drop rate: 99.97815533%

IPv4 - 131068 Clients - 100k invalid PPS - 50k valid PPS



IPv4 - 131068 Clients - 100k invalid PPS - 50k valid PPS

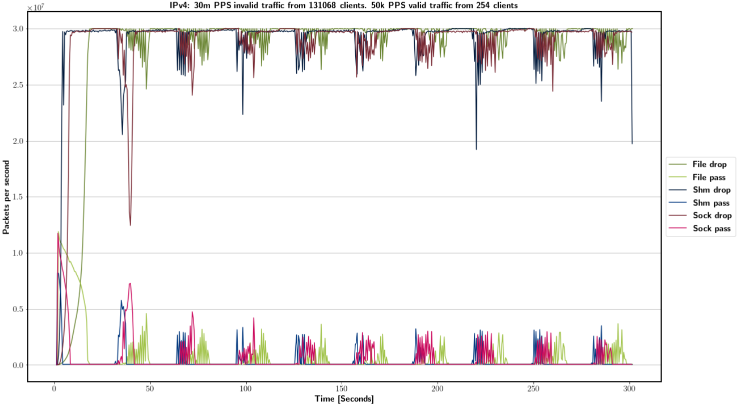
IPC type	XDP_DROP [10^6]	XDP_PASS [10^6]	Relative drop [%]
File	25.99	19.01	99.69409958
Shm	26.46	18.54	101.5083842
Sock	26.44	18.56	101.4395334

IPC type	Packets received by udp_server [10^6]	Log messages [10^6]	CPU [seconds]
File	18.16	3.54	08.34
Shm	18.54	3.54	10.14
Sock	18.53	3.54	100.40

Block

Total packets sent: 45m. Best-case drop rate: 86.8932%

IPv4 - 131068 Clients - 30M invalid PPS - 50k valid PPS



IPv4 - 131068 Clients - 30M invalid PPS - 50k valid PPS

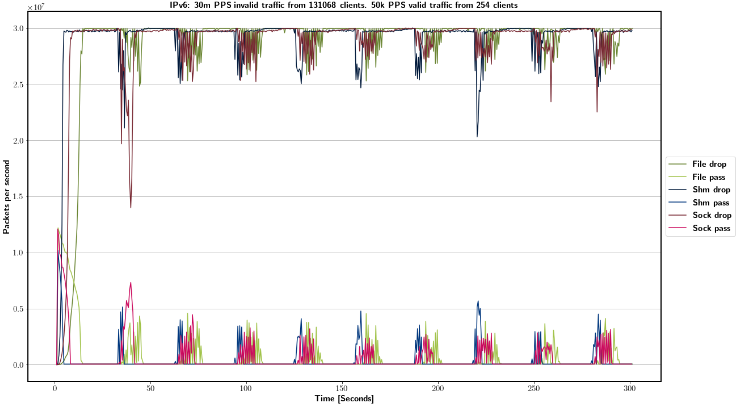
IPC type	XDP_DROP [10^8]	XDP_PASS [10^6]	Relative drop [%]
File	85.02	238.30	94.51036756
Shm	87.57	104.14	97.33826458
Sock	86.12	180.89	95.73084169

IPC type	Packets received by udp_server [10^6]	Log messages [10^6]	CPU [seconds]
File	18.04	7.44	38.99
Shm	25.32	11.50	71.92
Sock	18.33	5.93	323.02

Block

Total packets sent: 9,015m. Best-case drop rate: 99.95631067%

IPv6 - 131068 Clients - 30M invalid PPS - 50k valid PPS



IPv6 - 131068 Clients - 30M invalid PPS - 50k valid PPS

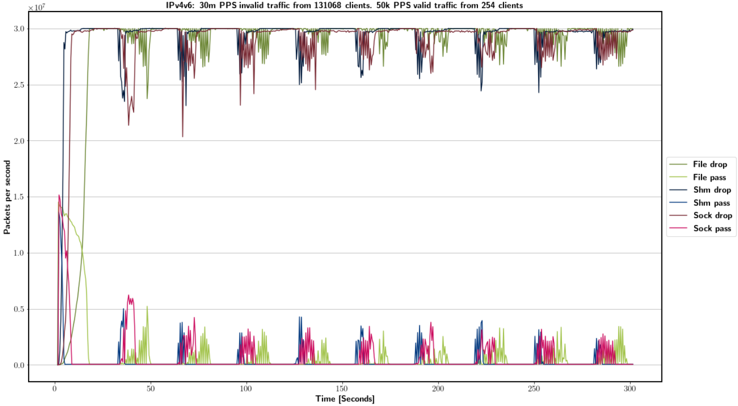
IPC type	XDP_DROP [10^8]	XDP_PASS [10^6]	Relative drop [%]
File	85.73	228.07	95.29278185
Shm	87.60	109.08	97.37706621
Sock	86.21	177.33	95.82614459

IPC type	Packets received by udp_server [10^6]	Log messages [10^6]	CPU [seconds]
File	17.90	6.91	38.41
Shm	25.08	11.15	74.71
Sock	18.67	6.18	317.37

Block

Total packets sent: 9,015m. Best-case drop rate: 99.95631067%

IPv4v6 - 131068 Clients - 30M invalid PPS - 50k valid PPS



IPv4v6 - 131068 Clients - 30M invalid PPS - 50k valid PPS

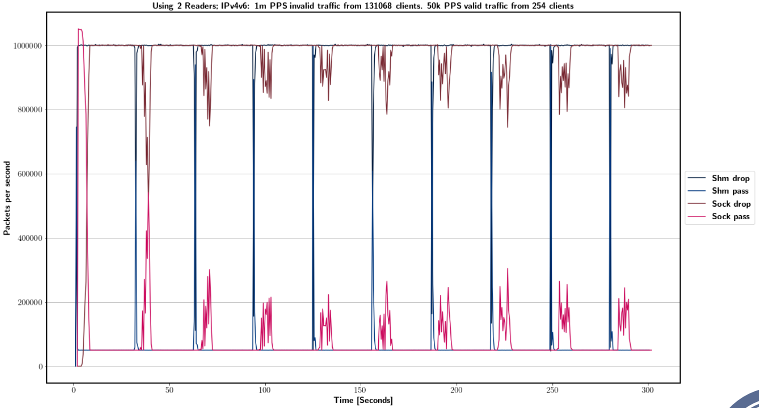
IPC type	XDP_DROP [10^8]	XDP_PASS [10^6]	Relative drop [%]
File	85.12	286.15	94.61335186
Shm	88.02	105.83	97.84149307
Sock	86.30	212.81	95.93428297

IPC type	Packets received by udp_server [10^6]	Log messages [10^6]	CPU [seconds]
File	17.69	7.07	47.15
Shm	25.13	11.16	94.64
Sock	18.00	5.99	353.34

Block

Total packets sent: 9,015m. Best-case drop rate: 99.95631067%

IPv4v6 2nd Reader - 131068 Clients - 1M invalid PPS - 50k valid PPS



IPv4v6 2nd Reader - 131068 Clients - 1M invalid PPS - 50k valid PPS

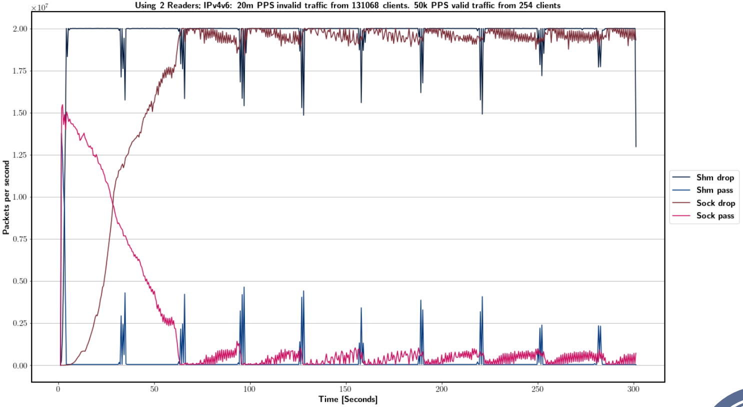
IPC type	XDP_DROP [10^7]	XDP_PASS [10^6]	Relative drop [%]
Shm	29.53	19.75	99.72283593
Sock	28.91	25.94	97.6334018

IPC type	Packets received by udp_server [10^6]	Log messages [10^6]	CPU [seconds]
Shm	19.48	4.49	17.76
Sock	18.29	4.15	80.82

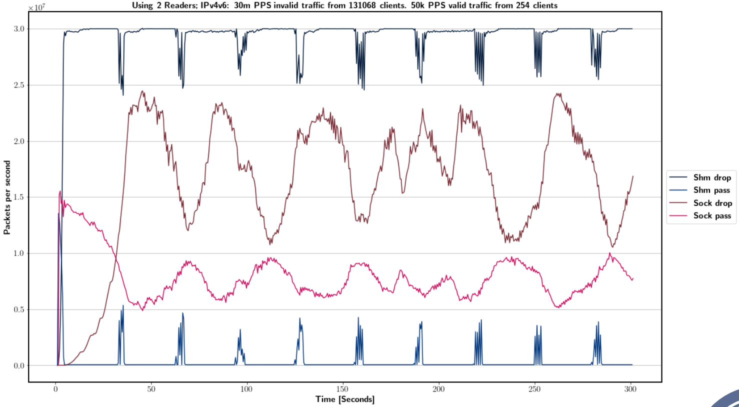
Block

Total packets sent: 9,015m. Best-case drop rate: 99.95631067%

IPv4v6 2nd Reader - 131068 Clients - 20M invalid PPS - 50k valid PPS



IPv4v6 2nd Reader - 131068 Clients - 30M invalid PPS - 50k valid PPS



Questions?

Questions?